Patent Application of

Zhe Li

For

METHOD FOR CONDITIONAL TAUTOLOGY CHECKING

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to provisional application No. 60/125835, titled Partial Tautology Checking Method Using Partial Enumeration and Simplification, and filed on March 24th, 1999, the contents of which are herein incorporated by reference..

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

Not Applicable.

REFERENCE TO A MICROFICHE APPENDIX

Not Applicable.

BACKGROUND OF THE INVENTION

This invention relates to Boolean functions~~, specifically to checking whether a Boolean function~~ is a tautology with a given constraint.

[illegible]

The value of a Boolean function can be either 0 or 1. A Boolean function can depend on any number of variables, and each variable's value can be either 0 or 1. The Boolean function maps each combination of values of these variables to either 0 or 1. A Boolean function is satisfiable if it maps at least one such combination to 1. A Boolean function is a tautology if it maps all combinations to 1. A Boolean function is a tautology if and only if the negation of the Boolean function is not satisfiable.

A tautology checking method decides whether a given Boolean function is a tautology. An equivalence checking method decides whether two given Boolean functions are equivalent. An equivalence checking method can be used as a tautology checking method if one of the two given Boolean functions is Boolean constant 1. A tautology checking method can perform equivalence checking if the given Boolean function is the exclusive nor (XNOR) of two Boolean functions.

Because the value of a signal in a digital circuit can be either 1 or 0, digital circuits are often represented as Boolean functions. Equivalence checking methods and tautology checking methods can be used to check whether two digital circuits are equivalent, which is used in functional design verification of digital circuits. The verification is generally performed as checking the equivalence between a specification and an implementation. These methods can be very useful when using computers to execute them.

~~As digital circuits become more and more complex while integrated circuit technologies grow, equivalence checking methods and tautology checking methods often fail to handle the complex Boolean functions used to represent digital circuits. The failures are either due to unreasonably long run times or due to requiring unreasonably large amounts of computer memory.~~ [illegible]

~~A way to avoid such failures is to check equivalence or tautology for only some of all combinations of the variable values. This is used widely in the industry because each of the ignored combinations is often similar enough to a combination involved in the checking or some combinations are less critical than others. This is normally performed as logic simulation for one combination at a time. When the same combinations are involved, logic simulation is usually much less efficient than equivalence checking methods or tautology checking methods.~~ [illegible]

There clearly are needs of more efficient methods for checking equivalence or tautology for a given subset of all combinations of the variable values.

BRIEF SUMMARY OF THE INVENTION

The present invention provides a method for determining whether a Boolean function is equivalent to Boolean constant 1 within a given subset of the input space.

The given subset is divided into a plurality of smaller subsets regardless how the smaller subsets are chosen. If any of the smaller subsets is not a cube, this smaller subset is divided further. If one of the smaller subsets is a cube, the Boolean function is simplified with constant substitution within the cube. If the simplification result is not a Boolean constant, the cube is divided further. If the simplification result is Boolean constant 0, a negative conclusion is reached. The conclusion is positive if none of the simplification result is Boolean constant 0.

Many of the subset division steps and many of the Boolean function simplification steps can be performed independently of one another, and therefore these independent operations can be performed separately at different times or on different computers. The given subset can expand or shrink dynamically if updates of the given subset are considered in the steps. These dynamic updates make it possible to dynamically adjust the divided subsets.

BRIEF DESCRIPTION OF DRAWINGS

FIG. 1 illustrates, in flow diagram form, a recursive process for conditional tautology checking.

FIG. 2 illustrates, in flow diagram form, an iterative process for conditional tautology checking.

FIG. 3 is a representative computer system in conjunction with which the embodiments of the present invention may be implemented.

DETAILED DESCRIPTION OF THE INVENTION

A method for automatically checking whether a Boolean function is equivalent to Boolean constant 1 with a given constraint, using a computer, is disclosed.

A Boolean function is a tautology if the Boolean function maps all points in the entire input space to Boolean constant 1. A tautology is equivalent to Boolean constant 1.

A Boolean function is a conditional tautology if the Boolean function maps all points in a given subset of the input space to Boolean constant 1, while it may map any point outside the given subset of the input space to either Boolean constant. A tautology is a conditional tautology when the entire input space is its given subset of the input space.

Conditional tautologies are easy to check because of the flexibility of doing divide-and-conquer with them. A Boolean function is a conditional tautology with a given subset of the input space if

(1) a number of smaller conditional tautologies all share this Boolean function, and

(2) these smaller conditional tautologies' given subsets of the input space jointly cover exactly the large conditional tautology's given subset of the input space

(i.e. the large conditional tautology's given subset of the input space is the union of these smaller conditional tautology's given subsets of the input space).

B3 [illegible]

~~A cube is a subset of the input space where some input variables are substituted with Boolean constant 1 and some other input variables are substituted with Boolean constant 0. The count of points in a cube must be 1, 2, 4, or any other power of 2 because zero or more input variables are completely free to take any values. A Boolean function is a conditional tautology with a cube as the given subset of the input space if the Boolean function is simplified to Boolean constant 1 after the input variables in the Boolean function are substituted with these 1's or 0's according to the cube's definition.~~

09534533.032400

B4 [illegible]

~~Conditional tautology checking can be performed in a recursive process as illustrated in FIG. 1. The process starts with a step **100** receiving a Boolean function and a given subset of the input space. A step **110** is performed next to determine whether to go to a step **120** or a step **150**. It must go to step **150** if the given subset of the input space is not a cube. It may go to either step **120** or step **150** under other conditions, but it must go to step **120** if the given subset of the input space includes only one point. The Boolean function is simplified in step **120** with substituting input variables in the Boolean function with Boolean constants according to the cube's definition. A step **130** is performed after step **120** to check the simplification result before going to step **150** or another step **140**. Step **150** is performed next if the simplification result is not a Boolean constant. Otherwise step **140** is performed next~~

to look at the Boolean constant resulted from the simplification. The positive conclusion is given in a step **180** after step **140** if the simplification result is Boolean constant 1. The negative conclusion is given in a step **190** after step **140** if the simplification result is Boolean constant 0. The negative conclusion may include at least one counter example indicating the substitution of the input variables with the Boolean constants.

Sub A4

In step **150**, the given subset of the input space is divided into several smaller subsets of the input space before moving on to a step **160**. For each of these smaller subsets of the input space, the conditional tautology checking process, starting from step **100**, is performed recursively for the Boolean function (the simplified copy if after step **120**) with this smaller subset as the given subset of input space. After all such recursions of step **160**, a step **170** is performed to summarize all conclusions collected from these recursions. If all the collected conclusions are positive, the positive conclusion is given in step **180**. If any of the collected conclusions is negative, the negative conclusion is given in step **190**.

The strategy of dividing a subset of the input space into smaller ones, in step **150**, can be arbitrary (manual or automatic), and a process of conditional tautology checking can use a mixture of several such strategies. Any strategy can get correct answer after a finite number of the recursive divisions because all smaller subsets of the input space cannot get smaller than the smallest cubes (also known as individual points of the input space). Because each of the smallest cubes has all input variables substituted with Boolean constants, it always gets the Boolean function simplified to a Boolean constant after a finite number of divisions.

In some cases, the strategy of dividing a subset of the input space, in step **150**, is to divide it into the largest cubes within the subset because this causes applying simplification, in step **120**, as early as possible. In some other cases, smaller cubes are preferred because large cubes have more chances to waste the simplification efforts in step **120**. A larger cube often has smaller chance to result in a Boolean constant from simplification with constant substitution in step **120**. These cubes may overlap with each other, and it depends on specific cases whether it is better to allow this overlapping or not.

~~Conditional tautology checking can also be performed in the following iterative process illustrated in FIG. 2. The iterative process starts with a step **200** receiving a Boolean function and a given subset of the input space. A step **210** is performed next to divide the given subset of the input space into two subsets. One is called a cube subset and the other is called a tail subset. The cube subset is a cube in the input space. If the given subset is a cube itself and it is chosen to be the cube subset, the tail subset is empty. A step **220** is performed after step **210** to simplify the Boolean function with substituting input variables in the Boolean function with Boolean constants according to the cube's definition. The simplification result can be used to replace the original Boolean function in all following steps and iterations if the tail subset is empty. Otherwise the original Boolean function is preserved for all~~

[illegible]

following steps and iterations, and the simplification result can be used temporarily as the Boolean ~~function to simply again in the next round, as described below, only~~ when the simplification result is not a Boolean constant.

[illegible]

A step **230** is then performed to check the simplification result before performing a step **240** or another step **250**. If the simplification result is not a Boolean constant, step **240** is performed to make the cube subset a smaller cube by dividing the given subset of input space similarly to in step **210**, and step **220** is performed again after step **240** for the smaller cube with the last simplification result as the Boolean function to simplify. Otherwise step **250** is performed next to look at the Boolean constant resulted from the simplification before performing a step **260** or another step **270**. The negative conclusion is given in step **260** if the simplification result is Boolean constant 0. The negative conclusion may include at least one counter example indicating the substitution of the input variables with the Boolean constants. If the simplification result is Boolean constant 1, step **270** is performed to examine whether the tail subset from step **210** or step **240** is empty. If the tail subset is empty, the positive conclusion is given in a step **290**. Otherwise a step **280** is performed making the tail subset the new given subset of the input space before looping back to step **210**.

There are many ways to represent a Boolean function in a computer for a program to process. This invention is not limited to any particular ~~representation~~ because the representation of Boolean functions is only involved in the simplification

[illegible]

~~and the constant~~ substitution within steps 120 and 220.

There are different ways to represent a subset of the input space. One way is to use a characteristic function. A Boolean function is a characteristic function of a subset of the input space if its value is 1 for and only for all points in the subset. Another way is to use ranges of binary integers.

Given an ordering of all the input variables, each point of the input space can be represented as a binary integer. Each bit of the binary integer represents the value of each input variable. The total count of these binary integers is the Nth power of two where N is the number of input variables. The lower bound of these binary integers is zero. A range of these binary integers represents a subset of the input space. A range has a lower bound and an upper bound, and it includes all binary integers between the two bounds (i.e. all binary integers that are greater than or equal to the lower bound and smaller than or equal to the upper bound). Any subset of the input space can be represented as one or more ranges of binary integers jointly.

A range of these binary integers represents a cube if the only difference between the range's upper bound and the range's lower bound is at the rightmost bits (i.e. the rightmost bits of the range's lower bound, if they are 0's, all correspond to 1's in the rightmost bits of the range's upper bound, respectively). A range from

10100 to 10101 represents a cube. A range from 10100 to 10110 does not represent a cube.

A7

Given any range, the range representing a cube that covers the first part of the given range can be easily identified because

(1) it shares the lower bound with the given range,

(2) its upper bound is obtained by turning the rightmost zero or more 0's of its lower bound into 1's, and

(3) its upper bound is smaller than or equal to the given range's upper bound.

09534533.032400

If several cubes satisfy these three conditions, one cube is selected depending on the chosen strategy. The selected cube divides the given range into two ranges: a range representing the cube and the other range representing the rest.

Based on the range representation of subsets of the input space, the conditional tautology checking method can be modified to handle a range whose upper bound is dynamically shifting. This shifting is useful because it enables dynamic division of a subset of the input space into several. The division criteria can be dynamically decided on-the-fly based on the changing factors of the computing environment. These changing factors can include any interrupts, memory shortage and time limit. It is the most interesting if the range is shrinking so that the cube used for simplification at the moment is completely outside the range: simply go to step

290 if it is in any other step. If the range is extending, the tail subset need to be extended to the upper bound of the range before step 270.

Furthermore, checking different conditional tautologies can be in different computing environment. For example, some smaller conditional tautologies can be moved to other computers to continue checking.

~~More generally, checking a conditional tautology can be performed as checking several smaller conditional tautologies concurrently if the division is decided in advance. Then each of the smaller conditional tautologies can be checked with dynamic divisions. Checking each conditional tautology can start from either end of the range representing the given subset of the input space. It can start from different ends for checking different conditional tautologies. If desired, the iterative process and the recursive process can be mixed in checking a conditional tautology. Different processes can be used either at different levels of the divisions or for different smaller subsets of the input space from the same division. When the (given or smaller) subset of the input space is a cube, any other tautology checking method other than simplification can also be applied after constant substitution.~~

[illegible]

~~With these flexible division techniques, very large Boolean functions can be handled in tautology checking. This is required for automatically verifying large data-processing systems using computers, especially when verifying them against the expected behaviors without any assumptions of their internal structures.~~

[illegible]

The invention discussed above may be implemented within dedicated hardware **15** as illustrated in FIG. **3** or within processes implemented within a data processing system **13**. A typical hardware configuration of a workstation, that may be implemented to accomplish the method disclosed herein, is illustrated and includes a central processing unit (CPU) **10**, such as a conventional microprocessor, and a number of other units interconnected via a system bus **12**. The workstation shown in FIG. **3** includes random access memory (RAM) **14**, read only memory (ROM) **16**, and input/output (I/O) adapter **18** for connecting peripheral devices, such as disk units **20** and tape units **40**, to bus **12**. A user interface adapter **22** is used to connect a keyboard device **24** and a mouse **26** to system bus **12**. Other user interface devices such as a touch screen device (not shown) may also be coupled to system bus **12** through user interface adapter **22**.

A communication adapter **34** is also shown for connecting the workstation to a data processing network **17**. Further, a display adapter **36** connects system bus **12** to a display device **38**. The method of the present invention may be implemented and stored in one or more of disk units **20**, tape drive **40**, ROM **16** and/or RAM **14**, or even made available to system **13** via network **17** through communication adapter **34** and thereafter processed by CPU **10**. Since the apparatus implementing the present invention is, for the most part, composed of electronic components and circuits known to those skilled in the art, circuit details will not be explained in any greater extent than that considered necessary as illustrated above, for the understanding and appreciation of the underlying concepts of the present invention and in order not to obfuscate or distract from the teachings of the present invention.

While the above invention has been described with reference to certain preferred embodiments, the scope of the present invention is not limited to these embodiments. One skilled in the art may find variations of these preferred embodiments that, nevertheless, fall within the spirit of the present invention, whose scope is defined by the claims set forth below.